risti
Revista Ibérica de Sistemas e Tecnologias de Informação
Revista Ibérica de Sistemas y Tecnologías de Información

Recebido/Submission: 08/01/2023
Aceitação/Acceptance: 06/04/2023

Sistemas Automáticos de Calificación de Tareas de Programación: Revisión y Crítica

Carlos Cares[1], Ricardo Gacitúa[1]

carlos.cares@ceisufro.cl; ricardo.gacitua@ufrontera.cl

[1] Departamento de Ciencias de Computación e Informática, Universidad de La Frontera, Temuco 4811230, Chile

DOI: 10.17013/risti.50.58–72

Resumen: La corrección de las tareas de programación de computadoras es una actividad crítica y de alto consumo de tiempo para los académicos. Como respuesta a este problema, se han propuesto diversos sistemas de retroalimentación y calificación automática. Este artículo presenta una revisión de sistemas y enfoques de corrección y calificación automatizada de tareas de programación con el objetivo de ofrecer un mapa general de las funciones de estos sistemas y sus desafíos. Los resultados muestran que existe una importante diferencia entre la industria y la academia, así como una carencia en marcos de evaluación para estas herramientas.

Palabras-clave: Programación; enseñanza; calificación automática; retroalimentación; resultados de aprendizaje.

Automatic Grading Systems for Programming Assignments: Review and Critique

Abstract: Reviewing programming assignments has become a time-consuming and critical activity for academics. Various feedback and automatic grading systems have been proposed in response to this problem. This article presents a review of various systems and automated feedback approaches for grading programming assignments to provide an overview of the functions of these systems and their challenges. The results show a difference between the tools proposed by the academy and those proposed by the industry, as well as a lack of evaluation frameworks,

Keywords: Programming; teaching; automatic grading; feedback; learning outcomes.

1. Introducción

La programación de computadores se trata de componer y organizar un conjunto de instrucciones en un lenguaje de programación con el objetivo de resolver un problema de proceso de información. El desarrollo de esta habilidad es frecuentemente percibido como complejo por los estudiantes dado que la habilidad es compuesta y jerárquica (Ismail et al., 2010).

Por otra parte, la habilidad de programación presenta alta demanda. Algunas universidades norteamericanas han doblado sus matrículas y se reconoce una férrea competencia entre la industria y las universidades por el contrato de doctores en informática. A este respecto se ha llegado a hablar de una demanda "desbordada", caracterizada además por otros síntomas como loterías de cupos y largas listas de espera para los cursos de ciencias de computación (Singer, 2019).

De esta manera, los cursos masivos en línea (MOOC -Massive Open Online Courses-) han comenzado a tener una alta relevancia, no sólo por la masividad de la respuesta, sino también debido a la pandemia de Covid-19, que ha aumentado y fortalecido la percepción de satisfacción por un proceso de enseñanza en línea (Bianchi et al., 2022; Blinkin, 2022).

En este contexto, la enseñanza de la programación de computadores requiere por parte de los profesores una alta dedicación a la corrección y calificación de las soluciones de programación de los estudiantes, lo que constituye una barrera relevante para la escalabilidad de los cursos. A su vez, los tiempos de demora impactan negativamente en la percepción de los estudiantes respecto de la calidad del curso (Ismail, 2021). Debido a esta necesidad histórica, pero además acentuada por la pandemia y aumento de la demanda, es que la asistencia automática en la corrección y calificación se hace relevante en la enseñanza de la programación (Daradoumis, 2013; Aldriye, 2019).

Adicionalmente, un tercer elemento es el desarrollo de las habilidades metacognitivas, lo cual requiere conciencia respecto de "qué es lo que sé", y del "cómo aprendo" (Armstrong, 1989). Las habilidades metacognitivas conforman parte de los objetivos de la enseñanza y, en particular, la corrección automática de programas, la identificación de errores y de desvíos en términos de métricas de programación, constituyen habilitadores para el autodiagnóstico estudiantil.

Un sistema de corrección automática (y calificación) de tareas de programación (CAP) podría incluir el reconocer qué programa respuesta contiene errores, si logra pasar todas las situaciones de prueba definidas, si la solución responde a todos los requerimientos de la tarea asignada, así como también si ofrece realimentación que habilite el aprendizaje. Los requerimientos para el producto podrían, por lo tanto, ser variados y complejos, y la forma de desarrollar y mostrar los resultados también. Poy y Gonzales-Aguilar (2014), muestran diversas debilidades del sistema como objetivos pedagógicos no alcanzados, y certificación a personas que no realizan las actividades interactivas. Ante la diversidad de logros y debilidades se ha llegado a afirmar que el problema siempre se ha abordado parcialmente (Hass, Yuan & Li, 2019).

El objetivo de este trabajo de investigación es analizar los sistemas automatizados de calificación, mostrar sus tipos y las herramientas existentes, tanto las propuestas por la academia como aquellas provenientes de la industria. Asimismo, establecemos una crítica técnico-pedagógica basada en las carencias que hemos percibido, y proponemos oportunidades de mejora.

La estructura del artículo es la que sigue, la sección 2 presenta los fundamentos básicos para caracterizar los sistemas de calificación automática de tareas de programación de computadoras. La sección 3 presenta la metodología general de la revisión y la base del enfoque de la investigación. La sección 4 presenta y caracteriza los resultados del estudio.

La sección 5 presenta una breve discusión respecto de las debilidades encontradas, así como del trabajo futuro.

2. Antecedentes

Debido al aumento en el uso de tecnologías digitales y la masificación de los cursos masivos en línea, se han introducido diversos cambios que afectan los procesos de enseñanza-aprendizaje. Uno de estos cambios se refiere a la evaluación automática de tareas.

Evaluar tareas de programación de computadores manualmente conlleva una serie de problemas, entre ellos la inexactitud de la evaluación (Douce, 2005), así como el esfuerzo y el tiempo requerido, especialmente en cursos con gran cantidad de estudiantes. La evaluación automática se ha convertido en un método que suple estos problemas permitiendo no sólo calificar los ejercicios de programación de los estudiantes, sino también entregar información sobre la calidad de las soluciones.

La naturaleza de la programación puede considerar constructores y lógicas diferentes para su solución. En efecto, diferentes constructores y lógicas pueden producir el mismo resultado bajo un desarrollo diferente en forma y fondo. Adicionalmente, el tipo de problema pedido podría fácilmente caer en categorías donde no existen límites finitos para comprobar una solución satisfactoria (Budd & Angluin, 1982). Esta situación teórica agrega dificultad para determinar la corrección de una tarea de programación, haciendo el proceso lento y, además, proclive a errores.

Hass, Yuan & Li. (2019) han resumido las ventajas y las influencias entre el proceso de enseñanza-aprendizaje y el trabajo de la evaluación automatizada. En términos de enseñanza-aprendizaje la autora reconoce que (a) la evaluación no sesgada, (b) la realimentación es oportuna, y (c) existe posibilidad de autoaprendizaje. A su vez, en relación a las tareas de evaluación reconoce una (a) mejor disponibilidad, (b) reducción de la carga de trabajo, (c) mejor distribución y asignación de tareas, (d) un sistema verificable de entrega de tareas, (e) posibilidad de análisis de código y verificación de manera previa a la entrega por parte de estudiantes además de la asistencia a profesores. En este artículo hemos separado las ventajas reconocidas diferenciando las perspectivas del estudiante y del docente, y hemos agregado aquellas ventajas relacionadas al almacenamiento de los elementos, como por ejemplo el reuso de comentarios y casos de prueba. La Figura 1 ilustra el resultado.

Desde el punto de vista de la ingeniería del software, un programa es correcto si produce la salida esperada para cada entrada posible. Sin embargo, las posibilidades son más amplias, de este modo mientras algunos sistemas sólo detectan errores en una solución, otros permiten explicar estos errores e incluso ofrecer sugerencias de posibles mejoras (Striewe, 2009).

Los errores lógicos, también llamados semánticos, se refieren a aquellos errores que, aún cuando el compilador no arroje errores, provocan que la salida del programa no sea la esperada. Esto puede deberse a diferentes factores: mala comprensión del problema, errores lógico-matemáticos, desborde de las representaciones numéricas finitas, entre otros.

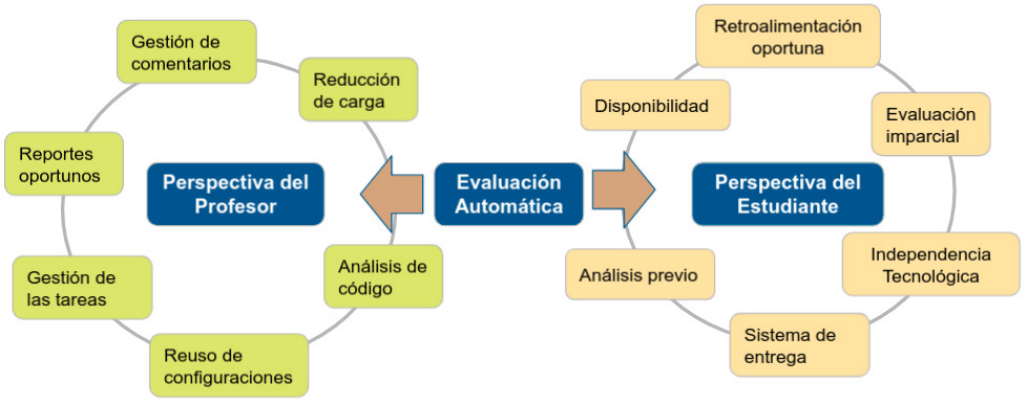

Figura 1 – Ventajas de la revisión automática de programas de computadora. Una extensión a la propuesta de Hass, Yuan & Li (2019)

Los errores de tiempo de ejecución aparecen cuando el programa está funcionando y lleva a que el programa deje de ejecutarse. Por ejemplo, debido a una comprobación de límites de la pila de sucesiones de invocaciones, excepciones de punteros a direcciones de memoria inaccesibles o inexistentes, fuentes de datos no encontradas o no disponibles, entre varios otros factores.

En términos generales, una herramienta automatizada de calificación puede considerar dos enfoques principales: (1) un análisis estático, donde el objetivo es analizar cómo está representada la solución en el correspondiente lenguaje de programación y (2) un análisis dinámico, donde el objetivo es analizar cómo se comporta la solución.

De este modo, el análisis estático funciona sin que se requiera la ejecución del código fuente, y sin embargo, se requiere del código para poder analizarlo. En el análisis dinámico, el código fuente se compila y se ejecuta mediante casos de prueba con diferentes datos de entrada para producir determinados datos de salida. Por definición, los sistemas capaces de realizar análisis dinámicos de código pueden realizar también análisis estáticos. Si la funcionalidad abarca estos dos tipos de análisis entonces son denominados sistemas híbridos.

3. Metodología

La verificación automática de programas de computador para propósitos de enseñanza académica continúa siendo un desafío, puesto que es posible evaluar programas simples automáticamente pero, para tareas más complejas, aún presentan limitaciones. Cuando se trata de la comprobación dinámica de errores, la situación es aún peor. Hass, Yuan & Li (2019) reconocen que para algunos casos se dispone de prototipos de investigación rudimentarios y sus prestaciones no se han probado a fondo ni han resultado satisfactorias. En la mayoría de los casos, sólo se permite un lenguaje de programación o un número limitado de tipos de trabajo.

Considerando, por tanto, la falta de claridad acerca de las características efectivas de estas herramientas automatizadas de calificación, y la carencia de evidencia que muestre el real uso de estas herramientas en la práctica, parece razonable establecer una caracterización de diferentes sistemas de calificación de estudiantes y presentar evidencia de su uso.

Ante la existencia de herramientas de carácter comercial y otras herramientas que son prototipos académicos, se utiliza una estrategia de revisión cualitativa de la literatura, la cual aborda la realidad desde una perspectiva holística e intenta comprenderla o describirla sin recurrir para ello a formular hipótesis, establecer medidas objetivas, o controlar exhaustivamente todas las variables, pero sí corresponde a una metodología de tradición crítica, exponiendo los hechos bajo calificaciones sociales (Sandín-Esteban, capítulo 7, 2003). Por el mismo motivo, este estudio no tiene pretensiones predictivas sino de generar los fundamentos y comunicación de las cualidades a considerar.

El objetivo de este trabajo de investigación es entonces describir las características principales de los sistemas automatizados de calificación de tareas de programación y recoger evidencia sobre el nivel de interés, como medida indirecta, de su uso. En término de las fuentes de datos, se recurre a bases de datos académicas y comunidades de usuarios de Internet para determinar las herramientas que son consideradas para su posterior análisis. En la Figura 2 ilustramos las fases del trabajo: los círculos son procesos y los rectángulos las entradas y salidas de información.

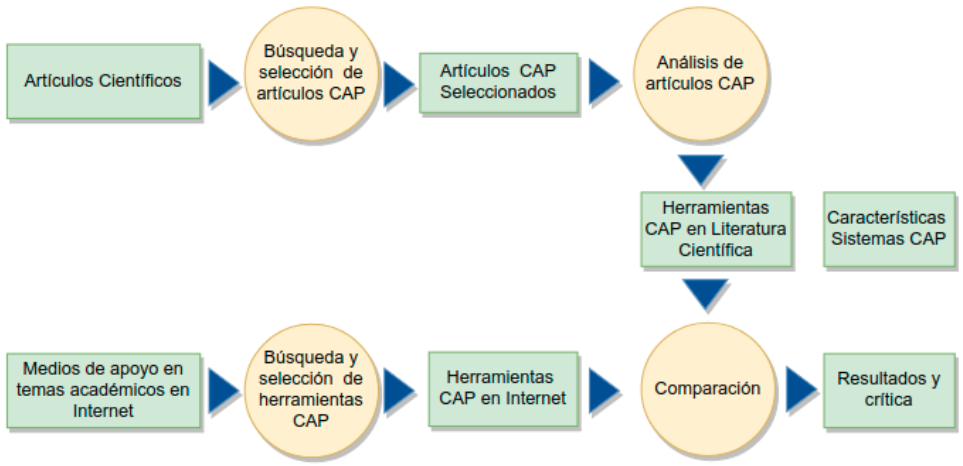

Figura 2 – Fases seguidas para generar la comparación y crítica

El protocolo descrito en la Figura 2 tiene por objeto permitir la descripción de características de las herramientas mencionadas en la literatura y determinar evidencias de uso. Adicionalmente, y considerando que Storey et al. (2014) plantean que existe un conocimiento colectivo en las redes sociales y comunidades de usuarios. Así, recopilamos evidencia por este medio. En este caso, se consideran dos fuentes de información, los foros de la red social Researchgate (https://www.researchgate.net), la cual es una red de

académicos y en cuyos foros es posible seguir el hilo de una conversación, de modo que las interacciones conforman una medida del interés en el tópico. La segunda fuente en Internet considerada es Youtube (https://www.youtube.com/), la cual es una red para compartir vídeos. En el caso de esta segunda red, es visible la cantidad de veces que un vídeo se ha activado, constituyendo así una medida de la popularidad del material disponible (Maraza-Quispe, 2020).

4. Ejecución del Estudio

4.1. Búsqueda y Selección de Artículos

La primera fase de nuestra revisión cualitativa de literatura comprende la definición de las fuentes de información. Considerando la existencia de algunas revisiones sistemáticas publicadas en la literatura, las cuales listan sistemas automatizados de calificación de tareas de programación, se considera como primera fuente de información este tipo de metaanálisis. La cadena de búsqueda se construyó a partir de la Tabla 1. Los términos dentro de una fila se conectan con "OR" y los términos dentro de la columna se conectan con "AND".

N°	Términos de Búsqueda
1	Revisión, Mapeo. Metaanálisis, estudio de casos, ensayo
2	Calificación, Evaluación
3	*Tarea, Asignación*
4	Programación

Tabla 1 – Términos de Búsqueda en la Revisión Bibliográfica

La cadena de búsqueda se aplicó en las bibliotecas digitales de IEEEXplore, ACM, y en la base de datos Scopus. Debido al bajo número de artículos ampliamos a una búsqueda general en Google Scholar. Estas son bases de datos conocidas y contienen información actualizada de artículos de revistas y conferencias disponibles a cualquier lector. Se revisaron las entradas para su adecuación e inclusión en el estudio. Se consideró, como criterio de inclusión, sólo revisiones sistemáticas, mapeos sistemáticos y metaanálisis. Los años considerados fueron aquellos entre 2015 y 2021. El resultado de la búsqueda permitió identificar cinco artículos, los cuales son presentados en la primera columna en la Tabla 2.

4.2. Análisis de los Estudios de Revisión

En el contenido de estos estudios se identificaron 52 sistemas de graduación automática o semi-automática. Sin embargo, en cinco casos, el sistema se identificó sólo por medio de los nombres de sus autores, es posible asumir entonces que estos productos no tienen un nombre o identificación comercial, o al menos no la tenían en el momento de la publicación. En 47 de estos estudios sí hubo un nombre de producto identificado. Las coincidencias las hemos resumido en la Tabla 2. De este modo es posible ver, por

ejemplo, que el producto “Web-Cat” es analizado en 4 de los 5 artículos seleccionados. El sistema denominado “OL Judge” es analizado en 3 de los 5 artículos. El resto de las coincidencias corresponden al mínimo, es decir, sólo 2 artículos mencionando el mismo producto. Los productos sobre los cuales coinciden los autores son sólo 8 de los 52. Cualitativamente podemos afirmar que tenemos una baja coincidencia entre los autores. Parte de esta diferencia podría estar en los años de los artículos, lo que indicaría que varios productos no tuvieron continuidad.

N°	Referencia	Autograder	AutoLEP	Boss	Mooshak	OL judge	Web-Cat	ASSYST	ASys
P1	Souza, Felizardo, & Barbosa (2016)	☑	☑	☑	☑	☑	☑	☐	☐
P2	Gupta and Gupta (2018)	☑	☑	☑	☑	☑	☑	☑	☐
P3	Aldriye , Alkhalaf & Alkhalaf (2019)	☐	☐	☐	☐	☐	☑	☑	☑
P4	Hass, Yuan & Li (2019)	☐	☐	☐	☐	☐	☑	☐	☐
P5	Ismail and Lakulu (2021)	☐	☐	☐	☐	☑	☐	☐	☑

Tabla 2 – Coincidencia en los productos analizados

4.3. Características de los Productos de Calificación Automática

En estos cinco estudios seleccionados se describen los productos de calificación automática productos de su análisis comparativo. En estas descripciones algunos categorizan las funcionalidades y otros simplemente describen los productos. El supuesto que realizamos es que las características descritas son aquellas que resultan relevantes para los académicos que realizan los estudios, de este modo, asumimos que las características no nombradas no se consideran relevantes.

La aplicación automática de pruebas es una característica transversalmente reconocida, pero, al mismo tiempo, la generación de estos casos de prueba no es considerada una característica para evaluar.

Otro tipo de características, clasificadas generalmente (pero no siempre) como análisis estáticos, se trata de patrones de comparación o métricas de código relacionadas a orientación a objetos (clases y métodos), a métricas de código, o calificación de patrones estructurales, por ejemplo que lo esperado como respuesta es un algoritmo con máximo una condición dentro de un ciclo. Otros elementos comunes a calificar son comentarios y estilos de codificación. En la Tabla 3 hemos resumido un conjunto de características reconocidas por los estudios.

4.4.Medios de Apoyo a la Academia

La relevancia de los medios de comunicación digital y de las redes sociales en particular, conforman un recurso para compartir dudas, y obtener respuestas. Storey et al. (2014) han reconocido la relevancia de blogs, foros, y otras redes sociales como fuente de información práctica, masiva y accesible. En la red Researchgate encontramos un foro con la pregunta: "What tools do you use for automated grading of assignments that involve programming?". Las respuestas van desde el año 2013 hasta el año 2017. Siete herramientas son mencionadas en las diferentes respuestas. Nos dimos a la tarea de buscarlas para verificar que aún existían. Sólo en un caso la herramienta no fue posible de recuperar. El resultado de los hallazgos los mostramos en la Tabla 4.

Funcionalidades Reconocidas		Souza, Felizardo, & Barbosa (2016)	Gupta and Gupta (2018)	Aldriye , Alkhalaf & Alkhalaf	Hass, Yuan & Li (2019)	Ismail and Lakulu (2020b)
Pruebas		☐	☑	☑	☑	☑
	Errores	☑	☑	☑	☑	☐
	Pruebas de software	☑	☑	☑	☑	☑
	Pruebas unitarias (cuestionarios)	☑	☑	☑	☐	☐
	Competencia (comparativa)	☑	☐	☑	☐	☐
	Generación de casos de prueba	☐	☐	☐	☐	☑
Rendimiento		☑	☐	☑	☑	☐
Patrones		☐	☐	☑	☑	☐
	Clases / Funciones esperadas	☐	☐	☑	☑	☐
	Estructura de Programas	☐	☐	☑	☐	☑
	Estructuras similares (plagio)	☑	☐	☐	☐	☑
	Estilo de programación	☑	☑	☐	☑	☐
Métricas		☐	☐	☐	☑	☐
	Código (líneas, complejidad)	☑	☐	☐	☑	☐
	Comentarios	☑	☑	☐	☑	☐
Retro-alimentación		☑	☑	☑	☑	☑
	Corrección inmediata	☑	☐	☑	☑	☐
	Oportunidad de corregir	☑	☐	☐	☑	☐
	Entre estudiantes	☐	☐	☑	☐	☐
	Profesor	☑	☑	☐	☐	☑
Historial		☑	☐	☑	☑	☐
	Gestión recursos profesor	☑	☐	☑	☑	☐
	Revisión detallada estudiante	☑	☑	☐	☐	☐

Tabla 3 – Características comunes en los productos CAP

Año	Respuestas a uso de calificación automática de programas
2013	Web-CAT
2015	Stepic (encontrado como Stepik)
2015	Fitchfork (no encontrado)
2016	Inginious (encontrado como ECI Inginious)
2016	Autolab

Año	Respuestas a uso de calificación automática de programas
2016	Autogradr (encontrado en csforall.org)
2016	Stepik
2017	Web-CAT

Tabla 4 – Respuestas de productos CAP en Researchgate

Una segunda fuente muy común de material educativo, particularmente en la oferta de vídeos, es Youtube (youtube.com). Si bien en esta red los vídeos son de todo tipo, también ha sido reconocida como una fuente habitual de material documental (Maraza-Quispe et al. 2020). En Youtube se hizo la búsqueda de vídeos tutoriales de los programas identificados y, además, se realizó una búsqueda por las palabras claves previamente identificadas. Para cada producto se buscaron los primeros cinco vídeos más vistos y sumamos la cantidad de visualizaciones. En la Tabla 5, hemos resumido estos resultados. En el caso del producto Stepik, hemos dejado dos celdas en amarillo debido a que dichos vídeos están en ruso. Todo el resto está en inglés. No se encontró nada en español.

4.5. Comparación

Hemos realizado una revisión de estudios comparativos de herramientas de calificación automática de tareas de programación. A su vez, hemos buscado en redes de asistencia académica como Researchgate y otra de fuentes documentales como Youtube, sobre recomendaciones y tutoriales.

Herramienta	Vídeo 1	Vídeo 2	Vídeo 3	Vídeo 4	Vídeo 5	Total	Porcentaje
Gradescope	69,021	14,030	12,512	12,227	7,455	115,245	59.6%
Nbgrader	16,861	1,739	1,679	1,177	1,142	22,598	11.7%
Github Classroom	7,839	6,376	2,932	2,345	1,376	20,868	10.8%
Uva Online Judge	4,850	3,706	3,082	2,605	2,200	16,443	8.5%
Web-Cat	2,017	2,006	1,886	546	444	6,899	3.6%
Codegrade	2,559	751	684	167	125	4,286	2.2%
Autolab	866	859	424			2,149	1.1%
Mooshak	767	371	364	339	284	2,125	1.1%
Stepik	829	447	280	223	177	1,956	1.0%
ECI Inginious	727	34	25	23	11	820	0.4%

Tabla 5 – Visitas en Youtube de tutoriales de productos CAP

Lo primero a destacar es que, dentro de los estudios, existe una baja coincidencia de las herramientas a considerar como parte del estudio. En todos los casos, las fuentes

de herramientas fueron otros estudios académicos donde se proponían herramientas CAP más que búsquedas en el mercado o repositorios de software. De los cinco estudios seleccionados, por ejemplo, no hay ninguna herramienta CAP que haya sido considerada en todos los estudios. Sólo una (Web-Cat) fue considerada en cuatro de los estudios seleccionados.

4.6. Comparación

Hemos realizado una revisión de estudios comparativos de herramientas de calificación automática de tareas de programación. A su vez, hemos buscado en redes de asistencia académica como Researchgate y otra de fuentes documentales como Youtube, sobre recomendaciones y tutoriales.

Lo primero a destacar es que, dentro de los estudios, existe una baja coincidencia de las herramientas a considerar como parte del estudio. En todos los casos, las fuentes de herramientas fueron otros estudios académicos donde se proponían herramientas CAP más que búsquedas en el mercado o repositorios de software. De los cinco estudios seleccionados, por ejemplo, no hay ninguna herramienta CAP que haya sido considerada en todos los estudios. Sólo una (Web-Cat) fue considerada en cuatro de los estudios seleccionados. La moda (valor más repetido) ha sido dos, es decir, lo habitual es que, entre los estudios académicos, la coincidencia es de sólo dos herramientas. En los estudios, existe sólo un 50% de coincidencia en las recomendaciones de foro, es decir, de las ocho herramientas CAP recomendadas en Researchgate, sólo cuatro habían sido mencionadas entre las 52 encontradas. En la Tabla 6 hemos resumido las coincidencias coloreando de verde las coincidencias entre tres fuentes y en amarillo las coincidencias entre dos.

Herramienta	Estudios (Total 5)	Recomendaciones (Total 8)	Visitas Youtube (Total 193,389)
Autograder	2/5	-	
AutoLEP	2/5	-	
ASys	2/5	-	
Boss	2/5	-	
Mooshak	2/5	-	2,125 (1%)
OL Judge	3/5	-	16,443 (9%)
Web-Cat	4/5	2/8	6,899 (7%)
ASSYST	2/5	-	
INGinious	1/5	1/8	820 (<1%)
Stepik	-	2/8	
Autolab	-	1/8	2,149 (2%)
Gradescope	-	1/8	115,245 (60%)
NBgrader	-	-	22,589 (23%)
Github Classroom	-	-	20,868 (21%)
Codegrade	-	-	4,286 (2%)

Tabla 6 – Coincidencias de herramientas

4.7. Crítica

El primer comentario que aparece obvio ante los resultados obtenidos es que estamos en frente de un área de estudio donde los objetos de estudio, es decir las herramientas de calificación automática, evolucionan rápidamente, y, como todo producto de software, se han movido los últimos años desde aplicaciones de escritorio a plataformas web, y también han evolucionado hacia el concepto de software como servicio. Sin embargo, muchas de las descripciones no hacen referencia básica a las características de los productos en cuanto a su plataforma tecnológica, ni forma de distribución o modelo de negocio si es que existe.

Adicionalmente, como parte de los hallazgos, varias de las herramientas mencionadas en los estudios no fueron encontradas en la revisión. Esto quiere decir que muy probablemente estas herramientas no tenían una versión para distribuir ni siquiera a nivel de prototipo experimental, menos como un producto o servicio. De este modo, el considerar un marco de referencia para la evolución desde prototipo a producto debería ser un requisito en la descripción de estos productos, pues no resulta comparable una idea inicial expresada en un prototipo a un producto maduro y con un equipo de desarrollo de software de tiempo completo. En este sentido, la propuesta original de la NASA para la evolución de los prototipos, que distingue nueve niveles, de TRL-1 a TRL-9, podría conformar una referencia conocida con este propósito (Olechowski, Eppinger & Joglekar, 2015).

Estos elementos particulares que hemos distinguido aquí, es decir, el tipo de tecnología de base, el modelo de negocio de la oferta del producto, y el estado de progreso, estabilidad o desarrollo del producto en sí, constituyen características de calidad de un producto de software. El conjunto de estas características, sus subcaracterísticas y métricas conforman lo que se denomina un modelo de calidad (Carvallo, Franch & Quer, 2006). Los modelos de calidad de un producto de software deberían estar basado en alguna referencia, como por ejemplo el estándar ISO/IEC 25000, que ha alcanzado un alto nivel de madurez como para considerarlo como referencia (Nistala, Nori & Reddy, 2019).

En este sentido, la Tabla 3, se representa una primera aproximación a la dimensión de calidad de la ISO/IEC 25010 denominada “Adecuación Funcional”. En esta dimensión de calidad, se debe considerar la calidad de completo, la calidad de correcto, y la calidad de pertinente. En particular, la calidad de pertinente, debería poder definirse y posteriormente calificarse sólo en la medida del correspondiente objetivo pedagógico. Selby (2015) caracteriza, dentro de la taxonomía de Bloom, las habilidades que pueden ser evaluadas mediante la realización de un programa, estás van desde la comprensión básica de instrucciones de programación (comprensión) hasta alta competencias cognitivas como la capacidad de síntesis y evaluación. Es decir, el objetivo pedagógico, que podría establecerse en relación a la taxonomía de Bloom, cubre un amplio espectro de competencias. De este modo, las funcionalidades de las herramientas deberían poder mapearse a los resultados de aprendizaje que conforman el contexto pedagógico. En todos los estudios considerados, aparece que la competencia de programación como una competencia atómica (indivisible). Sin embargo, como se ha mostrado con anterioridad (Fuller et al. 2007), la programación manifiesta ciertos niveles de competencia diferenciada, y por lo tanto, es esperable que exista una relación entre el nivel de

competencia a evaluar y las funcionalidades de las herramientas. Esta división no es sólo teórica, sino que se ha llevado a la práctica con éxito en una prueba de programación concreta (Costa y Miranda, 2017). Estas posibles funcionalidades y formas de evaluación deberían ser parte de un modelo de calidad.

Otra dimensión relevante en un modelo de calidad es la seguridad. La aplicación de calificación automática debería tener suficiente seguridad como para que un estudiante no pueda, a pesar que lo intente, verificar las respuestas de otros estudiantes o los ejemplos de respuestas correctas provistos si existiesen. Una característica no funcional, como la seguridad, puede tomar forma en funciones no observables (como la encriptación), o puede tomar formas observables, como la implementación de una particular política de actualización y propiedad de las claves. Estas alternativas de calidad deberían también formar parte de las subcaracterísticas de calidad en un modelo de calidad difundido y compartido como el propuesto.

5. Conclusiones

En este artículo, hemos revisado los trabajos sobre programas que realizan una calificación automática de las tareas de programación de computadores. Hemos buscado estudios de revisión de estos tipos de programas y hemos encontrado cinco. Entre los cinco estudios, que van entre 2016 y 2021, se nombran un total de 52 aplicaciones. Alternativamente, hemos buscado en redes sociales para encontrar otras aplicaciones usadas. Hemos encontrado en la red académica Researchgate un foro de recomendaciones sobre aplicaciones de este tipo. La tercera fuente analizada fueron los vídeos tutoriales en Youtube.

El principal hallazgo es la baja coincidencia entre los tipos de estudios y las menciones en las redes sociales. En el caso de los estudios, además, hay una baja coincidencia entre ellos. La principal conclusión de estos hallazgos es que los estudios académicos consideran como fuente otros estudios académicos y productos de los cuales no existe antecedentes como para poder afirmar que hayan alcanzado algún nivel de madurez tecnológica.

Un segundo hallazgo es que las características que los estudios destacan, y las funcionalidades analizadas, no tiene un nivel de detalle y tampoco una referencia que permita poder establecer una comparación repetible y usable también por otras herramientas del mismo tipo.

En el contexto anterior, hemos establecido una crítica a esta área de estudios. La crítica tiene un eje pedagógico, y un eje tecnológico. El eje pedagógico dice relación con la baja conceptualización de la habilidad de programación y su relación a un contexto de resultados de aprendizaje bajo alguna gradualidad de progreso ampliamente aceptada, como por ejemplo la taxonomía Bloom. El segundo elemento de la crítica es el eje tecnológico. Hemos formulado que el conjunto de características a evaluar debería estar guiado por algún estándar de calidad, y hemos afirmado que el ISO/IEC 25.000 es adecuado

La principal limitación de este estudio, es que no hemos intentado levantar un catastro contemporáneo de las herramientas y servicios de calificación automática de tareas

de programación. Por lo tanto, el estudio se basa en descripciones de terceros, y cuando revisamos y resumimos las características, es un esfuerzo de síntesis sobre las características que otros han reconocido.

Con la limitación descrita, la contribución que se realiza, es una contribución al cómo se puede progresar en esta área de estudio, esto incluye (1) establecer una relación teórica entre el progreso cognitivo y habilidades de programación de computadores de los estudiantes, y las funcionalidades de la aplicación y (2) establecer un modelo de calidad que incluya no sólo los aspectos funcionales sino una gama de atributos de calidad ya reconocidos en estándares de software.

Referencias

Aldriye, H., Alkhalaf, A., & Alkhalaf, M. (2019). Automated grading systems for programming assignments: A literature review. International Journal of Advanced Computer Science and Applications, 10(3). https://doi.org/10.14569/IJACSA.2019.0100328

Armstrong, A. M. (1989). The development of self-regulation skills through the modeling and structuring of computer programming. Educational Technology Research and Development, 37(2), 69-76. https://doi.org/10.1007/BF02298291

Bianchi, I. S., Daehn, C. M., Dávila, G. A., Tovma, N., & Shurenov, N. (2022). Cursos online Abertos Massivos (MOOCs) como potencializadores do conhecimento em Instituições de Educação Superior. RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação, (48), 59-73. https://doi.org/10.17013/risti.48.59–73

Blinkin, C. (2022, January 1). Dozens of U.S. colleges switching back to online classes as COVID-19 cases soar. Global news. https://globalnews.ca/news/8483197/ covid-us-colleges-online-classes/

Budd, T. A., & Angluin, D. (1982). Two notions of correctness and their relation to testing. Acta informatica, 18(1), 31-45. https://doi.org/10.1007/BF00625279

Carvallo, J. P., Franch, X., & Quer, C. (2006). Managing non-technical requirements in COTS components selection. In 14th IEEE Int. Requirements Engineering Conf. (RE'06) (pp. 323-326). IEEE. https://doi.org/10.1109/RE.2006.40

Costa, J. M., & Miranda, G. L. (2017). Desenvolvimento e validação de uma prova de avaliação das competências iniciais de programação. RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação, (25), 66-81. https://doi.org/10.17013/risti.25.66-81

Daradoumis, T., Bassi, R., Xhafa, F., & Caballé, S. (2013). A review on massive e-learning (MOOC) design, delivery and assessment. In 2013 8th Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing (pp. 208-213). IEEE. https://doi.org/doi 10.1109/3PGCIC.2013.37

Douce, C., Livingstone, D., & Orwell, J. (2005). Automatic test-based assessment of programming: A review. Journal on Educational Resources in Computing, 5(3), 4. https://doi.org/10.1145/1163405.1163409

Fuller, U., Johnson, C. G., Ahoniemi, T., Cukierman, D., Hernán Losada, I., & Jackova, J., et al. (2007). Developing a computer science specific learning taxonomy. In Proc. of the 7th Conf. on Innovation and Technology in Computer Science Education (pp. 152-170). United Kingdom: ACM. https://doi.org/10.1145/1345375.1345438

Gupta, S., & Anamika G.. (2017). E-Assessment Tools for Programming Languages: A Review. In Proc. of the 1st Int. Conf. on Information Technology and Knowledge Management (ICITKM'2017), New Delhi, India,(pp. 65-70).

Hass, B., Yuan, C., & Li, Z. (2019). On the automatic assessment of learning outcomes in programming techniques. In 2019 IEEE 14th Int. Conf. on Intelligent Systems and Knowledge Engineering (ISKE) (pp. 274-278). IEEE. https://doi.org/10.1109/ISKE47853.2019.9170370

Ismail, M. H., & Lakulu, M. M. (2021). A Critical Review on Recent Proposed Automated Programming Assessment Tool. Turkish Journal of Computer and Mathematics Education, 12(3), 884-894. https://doi.org/10.17762/TURCOMAT.V12I3.799

Ismail, M. N., Ngah, N. A., & Umar, I. N. (2010). Instructional strategy in the teaching of computer programming: a need assessment analyses. The Turkish online Journal of Educational Technology, 9(2), 125-131.

Maraza-Quispe, B., Alejandro-Oviedo, O., Fernández-Gambarini, W., Cisneros-Chavez, B., & Choquehuanca-Quispe, W. (2020). Análisis de YouTube como herramienta de investigación documental en estudiantes de educación superior. Publicaciones, 50(2), 133-147, https://doi.org/10.30827/publicaciones.v50i2.13949.

Nistala, P., Nori, K. V., & Reddy, R. (2019, May). Software quality models: A systematic mapping study. In 2019 IEEE/ACM Int. Conf. on Software and System Processes (ICSSP) (pp. 125-134). IEEE. https://doi.org/10.1109/ICSSP.2019.00025.

Olechowski, A., Eppinger, S. D., & Joglekar, N. (2015). Technology readiness levels at 40: A study of state-of-the-art use, challenges, and opportunities. In 2015 Portland Int. Conf. on management of engineering and technology (pp. 2084-2094). IEEE. https://doi.org/10.1109/PICMET.2015.7273196.

Poy, R., & Gonzales-Aguilar, A. (2014). Factores de éxito de los MOOC: algunas consideraciones críticas. RISTI - Revista Ibérica de Sistemas e Tecnologias de Informação, (E1), 105-118. http://doi.org/10.4304/risti.e1.105-118

Sandín-Esteban, M.P. (2003). Investigación Cualitativa en Investigación en Educación: Fundamentos y Tradiciones. Madrid; Mc Graw and Hill.

Selby, C. C. (2015). Relationships: computational thinking, pedagogy of programming, and Bloom's Taxonomy. In Proc. of the workshop in primary and secondary computing education (pp. 80-87). http://doi.org/10.1145/2818314.2818315.

Souza, D. M., Felizardo, K. R., & Barbosa, E. F. (2016). A systematic literature review of assessment tools for programming assignments. In 2016 IEEE 29th Int. Conf. on Software Engineering Education and Training (CSEET) (pp. 147-156). IEEE. http://doi.org/10.1109/CSEET.2016.48

Striewe, M., & Goedicke, M. (2009). Effekte automatischer Bewertungen für Programmieraufgaben in Übungs-und Prüfungssituationen. DeLFI 2009-7. Tagung der Fachgruppe E-Learning der Gesellschaft für Informatik eV.

Storey, M.A., Singer, L., Cleary, B., Figueira Filho, F., Zagalsky, A., (2014). The (r) evolution of social media in software engineering. In Proc. of Future of Software Engineering (FOSE), 100-116. https://doi.org/10.1145/2593882.2593887.